Conversational Chatbot

Winter in Data Analytics (WiDS)

Tanisha Khandelwal (190100127)

Mentor: Shravya Suresh

Introduction

The project is a part of Winter in Data Analytics (WiDS) conducted by Analytics Club. It provided a great opportunity to delve deep into the domains of Data Analytics, Machine Learning and AI and apply the knowledge and skills acquired throughout the course of the project to gain hands-on experience using a real-world application. Project introduces to the fundamentals of Natural Language Processing (NLP) and understanding its implementation in a Conversational Chatbot.

Following is the Week-wise project flow:

Week 1

Week 1 started with covering the basics in the domain of data science and revising and brushing up some of the previously learned concepts. This week content was crucial in building up the foundation for the entire duration of the project.

For this, referred to the YouTube video provided by Edureka which covered all the major concepts of Machine Learning, Statistics and Probability. This video was quite helpful in revising the concepts and also provided detailed insight into the topics.

Some of the topics covered in the video are:

- Different types of Machine Learning algorithms- Supervised, Unsupervised and Reinforcement Learning
- Regression and Classification Algorithms
- Decision Trees and how by combining several decision trees and using the concept of Ensemble Learning (i.e combining multiple weak learners to form a stronger model which would have better accuracy and yields better performance) we get Random Forest model.
- Detailed analysis of algorithms like Support vector Machine (SVM), KNN algorithm, Naves bayes, Clustering algorithms like K means clustering, Hierarchical clustering etc.

Week 2

This week's material was majorly focussed on covering the theoretical part which would be essential for the project. Started by learning with the basic terminologies and studied the various NLP concepts and theory that they will be required for the chatbot, such as Tokenization, Bag of Words etc. and other notions that will go into the intelligence of the Chatbot. Also, got introduced to N-Gram language models and understanding Naive Bayes classification and Sentiment Analysis.

Basic terminologies

- **Corpus**: A computer-readable collection of text or speech. It can be a single document or a collection
- **Lemma**: Set of lexical forms having the same stem, the same major part-of-speech, and the same word sense.
- **Regular Expressions**: It is used to specify text search strings, when we have a pattern to search for and a corpus of texts to search through. A regular expression search function will search through the corpus, returning all texts that match the pattern. A search can be designed to return every match on a line, if there are more than one, or just the first match.
- **Tokenization**: Separating out words from running text
- **Lemmatization**: Task of determining that two words have the same root, despite their surface differences. For example, the words sang, sung, and sings are forms of the verb sing. The word sing is the common lemma of these words, and a lemmatizer maps from all of these to sing.
- **Stemming**: Refers to a simpler version of lemmatization in which we mainly just strip suffixes from the end of the word.
- **Text Normalization**: Converting text to a more convenient, standard form
  Before almost any natural language processing of a text, the text has to be normalized. At least three tasks are commonly applied as part of any normalization process:
  1. Tokenizing (segmenting) words
  2. Normalizing word formats
  3. Segmenting sentences
- **Edit distance**: A metric used To compare words and other strings and measures how similar two strings are based on the number of edits (insertions, deletions, substitutions) it takes to change one string into the other.
- **Token**: Total number N of running words

N-gram language model

Models that assign probabilities to sequences of words are called language models or LMs. 2-gram (called bigram) is a two-word sequence of words and a 3-gram (a trigram) is a three-word sequence of words. n-gram models can be used to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire sequence. It is used to mean either the word sequence itself or the predictive model that assigns it a probability.

Represent a sequence of N words either as $w_1 \ldots w_n$ or $w_{1:n}$

P(w|h) = probability of a word w given some history h

$P(w_1,w_2,\ldots,w_n)$ = joint probability of each word in a sequence having a particular value $P(X = w_1, Y = w_2, Z = w_3, \ldots, W = w_n)$

Applying the chain rule of probability to words:

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned}$$

It shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.

The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words. The bigram model approximates the probability of a word given all the previous words $P(w_n|w_{1:n-1})$ by using only the conditional probability of the preceding word $P(w_n|w_{n-1})$. This can be generalized to the n-gram model which looks n−1 words into the past. Thus, the general equation for this n-gram approximation to the conditional probability of the next word in a sequence is

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

To estimate these bigram or n-gram probabilities we can use an intuitive way called maximum likelihood estimation or MLE. We get the MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1. For the general case of MLE n-gram parameter estimation:

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}\ w_n)}{C(w_{n-N+1:n-1})}$$

Evaluating Language Models

- The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called extrinsic evaluation. Extrinsic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand. Unfortunately, running big NLP systems end-to-end is often very expensive.
- An intrinsic evaluation metric is one that measures the quality of a model independent of any application. For this we need a test set.
- The probabilities of a n-gram model come from the corpus it is trained on, the training set or training corpus. Training corpus must have a similar genre to whatever task we are trying to accomplish and it should be in appropriate dialect or variety. The quality of a n-gram model can then be measured by its performance on some unseen data called the test set or test corpus.
- Test sets and other datasets that are not in our training sets are called held out corpora because we hold them out from the training data.
- If we are given a corpus of text and want to compare two different n-gram models, we divide the data into training and test sets, train the parameters of both models on the training set, and then compare how well the two trained models fit the test set. Whichever model assigns a higher probability to the test set—meaning it more accurately predicts the test set—is a better model.
- Raw probability is not used as a metric for evaluating language models, but a variant called perplexity. The perplexity (sometimes called PP for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words. For a test set $W = w_1w_2 \ldots w_N$,:

$$\begin{aligned}\mathrm{PP}(W) &= P(w_1w_2\ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1w_2\ldots w_N)}} = \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i|w_1\ldots w_{i-1})}}\end{aligned}$$

Minimizing perplexity is equivalent to maximizing the test set probability according to the language model. In computing perplexities, the n-gram model P must be constructed without any knowledge of the test set or any prior knowledge of the vocabulary of the test set. Any kind of knowledge of the test set can cause the perplexity to be artificially low. The perplexity of two language models is only comparable if they use identical vocabularies.

Naive Bayes

- Text categorization is the task of assigning a label or category to an entire text or document.
- Sentiment analysis: Extraction of sentiment, the positive or negative orientation that a writer expresses toward some object.
- Generative classifiers like naive Bayes build a model of how a class could generate some input data. Given an observation, they return the class most likely to have generated the observation.
- We represent a text document as if it were a bag-of-words, that is, an unordered set of words with their position ignored, keeping only their frequency in the document.
- Naive Bayes is a probabilistic classifier, meaning that for a document d, out of all classes c ∈ C the classifier returns the class ĉ which has the maximum posterior probability given the document.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

- We compute the most probable class ĉ given some document d by choosing the class which has the highest product of two probabilities: the prior probability of the class P(c) and the likelihood of the document P(d|c).
- Naive Bayes classifiers make two simplifying assumptions: The first is the bag of words assumption. Thus we assume that the features $f_1, f_2, \ldots, f_n$ only encode word identity and not position. The second is commonly called the naive Bayes assumption: this is the conditional independence assumption that the probabilities $P(f_i|c)$ are independent given the class c and hence can be ‘naively’ multiplied
- Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid underflow and increase speed

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

Training the Naive Bayes Classifier

- $P(w_i|c)$ is computed as fraction of times the word w_i appears among all words in all documents of topic c. We first concatenate all documents with category c into one big “category c” text. Then we use the frequency of w_i in this concatenated document to give a maximum likelihood estimate of the probability. Applying add-1 smoothing,

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} (count(w, c) + 1)} = \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

- Vocabulary V consists of the union of all the word types in all classes, not just the words in one class c
- Unknown words present in the set are ignored, they are removed from the test document and any probability for them is not included at all.
- Some systems choose to completely ignore another class of words: stop words, very frequent words like the and a.

Week 3

Began with the coding of chatbot. Referred to several resources and documentations available online to get familiarized with the TensorFlow framework covering some of the basic to more advanced topics.

Week 4

Completed the code for the implementation of the Conversational Chatbot.

This week also got introduced to Transformers which are quite popular and extensively used these days and understanding their function, working, architecture and encoder, decoder, sequence-to-sequence models. Using the pipeline function from Transformer library for approaching various NLP tasks like classifying sentences, text generation, extracting answer, mask filling, translation, summarization etc. Also looked at tokenization, conversion to input IDs, padding, truncation, attention masks and basic building blocks of a Transformer model.

Link to the github repository containing code for the project: https://github.com/tanisha605/WiDS-NLP-Chatbot/blob/main/Conversational_Chatbot_190100127.ipynb

Conclusion

Though I had some insight in Natural Language Processing (NLP) through deep learning specialization course but I wanted to do a project which would deal with the applications of NLP and would enable me to use its concepts and see what all domains can be explored and what all applications it can be useful. I believe that this project has really helped me to revise all the concepts that I have learnt and introduce me to a real-world problem where this can be utilized. This project has definitely added a new dimension to my learnings and helped me delve more into the domain of NLP.

References

Some of the resources referred to while working on the project are:

- Edureka's 10 hour data science course on YouTube: https://www.youtube.com/watch?v=-ETQ97mXXF0
- 'Speech and Language Processing' book by Daniel Jurafsky : https://web.stanford.edu/~jurafsky/slp3/
- TensorFlow Tutorial: https://www.youtube.com/watch?v=hvgnX1gbsLA&list=PLqnslRFeH2Uqfv1Vz3DqeQfy0w20ldbaV, https://www.tensorflow.org/api_docs/python/tf
- Transformers Material: https://huggingface.co/course/chapter1/1?fw=tf